Chapter 9.2 - Finite Difference Methods: Iterative Solution Methods for Ax=b

6 MIN · YOUTUBE · HTTPS://WWW.YOUTUBE.COM/WATCH?V=Y3K5JXFNOC8
https://www.youtube.com/watch?v=y3K5JxfnoC8

SUMMARY

The discussion focuses on efficiently solving the large-scale linear system \(Ax = b\) arising from the discretization of the shallow water equations. Given the impracticality of traditional methods like Gaussian elimination for large matrices, the speaker emphasizes the need for faster iterative algorithms to handle the computational demands of high-dimensional problems.

- *The problem involves solving \(Ax = b\) for large matrices generated from discretized shallow water equations.*
- *Traditional Gaussian elimination has a computational cost of \(O(n^3)\), making it infeasible for large systems (e.g., \(10,000 \times 10,000\) or \(1,000,000 \times 1,000,000\)).*
- *Efficient solutions require iterative methods rather than direct solvers to reduce computation time.*
- *Iterative schemes, such as Jacobi iteration, allow for progressively refining solutions by using previous estimates.*
- *These methods are particularly useful in high-dimensional computing, where initial guesses are close to the final solution.*
- *Advanced iterative techniques like generalized method of residuals and conjugate gradient methods are commonly used in large-scale climate and weather models.*
- *The speaker stresses the importance of adopting faster algorithms to avoid computational intractability in scientific computing.*

So we want to talk about ax equal to b at scale. And so one of the things that we've seen in the discretization of the shallow water equations is that we have a very large scale axis b problem that we've got to solve. And so part of the way we have to think about this is not just simply that I could just use a simple solver and do galaxy elimination. If the cost of that algorithm is order n cubed and n is, you know, a million grid points, you're looking at a really slow way to solve it. In fact, you can't in many practical applications if you try to solve it that way, it just wouldn't work. You don't have the memory. So, you need to start thinking about what are the fast algorithms available for me to solve ax= b at scale. I want to remind you of the problem we're trying to solve. The problem we're trying to solve is for the string function given the vorticity. And this is really important if we're going to advance the solution in time for so given the vorticity compute a string function and then we can evolve forward with the time stepping itself. But the ax equal to b portion is the most expensive part of the solution technique. So we have to really think about a good way to solve that right away. So this is what happened under discretization. I just want to remind you of organization of the data. We went ahead and discretized the stream function into n and m which are the spatial locations in x and y which came out with this equation here for each point. So it's using neighbors and point in front point behind in x point in front point behind in y minus twice the current point that's where you get this minus4 which is a contribution of two x deriv and two y derivatives and these neighboring points with boundary conditions. This led to a system of

differential equations. So if I discretize with a 100 points in X and 100 points in Y, then my system size, right, is going to be a 10,000x 10,000 system of equations.

So you can see the scaling of this number of equations goes up rather quickly because that's not even that big, 100 by 100 points. And yet now I've got a a matrix is going to be a 10,000 by 10,000 which is this matrix right here. So once you organize your data this is what you got to solve is ax equal to b. And so if I wanted to resolve more a thousand by a thousand grid points you know spatial discretization x and y all of a sudden I'm solving you know a million by a million matrix that I've got to do and this is very slow because the solution to this matrix goes like order n cubed.

So if this is a million-dimensional matrix, it goes like a million cubed as the operation cost. This is an impractical way to solve it. And so a lot of scientific computing is concerned with how do I bring the computation time down. Okay. And so I want to bring some methods to the table that people have thought about for a long time and that are actually in practical use every day in large scale computing. So first thing I want to do is revisit this here which is this is ax equal to b. Here's essentially the relationship that's for each set of equations. And so what I'm going to do with this is I'm going to be inspired. We've already looked at how to solve ax equal to b.

One of the things that you can do with solving x= b is you can come up with iterative schemes. very simple iterative schemes to say plug in a solution iterate to or plug in a guess try to iterate to a solution and so this is actually a construction of an iterative scheme which you can say look what I'm going to solve for here is each point I'm going to set up an iteration scheme so my point my next point in the iteration scheme the k + one is equal to all I did here is moved this term over which is the diagonal term over to the other side bring the delta squared over to here and just start iterating.

So we know that this is guaranteed to converge if it was strictly diagonal dominant. And this is a borderline case. It's strictly diagonal dominant is if all the all the terms on the off diagonal are less than the diagonal, but here they're equal. Okay, so we're on a marginal case. And so what we want to do though is figure out how to solve this. And here's an update rule. And this is very much inspired by Jacobe iteration schemes. And so in fact iterative schemes dominate a lot of really highdimensional computing problems because they actually if you're as you're marching forward into the future your guess for the future point is the current point. So you're not that far away from the solution. So the presumption is that it doesn't take very many iterations to get the update of the solution a delta t into the future. And so that's exactly what's used in CERN's big climate and weather models is methods like this iteration schemes and the ones that they use would be something like we'll talk about them is generalized method of residuals or by conjugate stabilized method which are sort of advanced iteration schemes for solving these problems. And so again, I just want to highlight that all I'm going after here is when you go to scale, if you're simply solving this by saying I'll use Gaussian elimination, you know, or if you're in mat lab a backslash or you just say np, you know, solve in in Python. It's really the wrong way to do it. You're paying way too much and you need to think about much faster techniques when you go to scale. Otherwise, you you know, you're just you can't do it. It's just it's order n cubed to solve that system and you'll very get in very quickly get in a situation where it's just computationally intractable if you're using something so slow. So everybody uses fast poison solvers and iteration techniques are one of the things that are available to you and

we'll talk about the code base for it but it's just based upon setting up an iteration structure to try leverage that to get a solution faster.